2

1992 Summer SIGAda AIWG Workshop Applications Experiences and Lessons Learned Panel Message From the Panel Chair

M92B0000103
September 1992

Janet Faye Johns

MITRE
Bedford, Massachusetts

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
September 1992

3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
1992 Summer SIGAda AIWG Workshop Applications Experiences and Lessons Learned Panel Message From the Panel Chair

5. FUNDING NUMBERS

6. AUTHOR(S)
Janet Faye Johns

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420

8. PERFORMING ORGANIZATION REPORT NUMBER
M92B0000103

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
same as above

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
same as above

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(Maximum 200 words)*
On 24-27 June 1992, the Association for Computing Machinery (ACM) Special Interest Group for Ada (SIGAda) Artificial Intelligence Working Group (AIWG) held a workshop to discuss Ada real-time and AI issues. Panels organized for the workshop focused on blackboard architectures, experiences and lessons learned, real-time development approaches and issues, and Ada 9X issues for AI systems. This paper assesses the state-of-the-art for AI development processes and provides a summary of the related workshop discussions from the experiences and lessons learned panel. Issues associated with requirements analysis, design methodologies, development techniques, test and validation, and maintainability are discusses for AI applications and for AI with Ada applications.

14. SUBJECT TERMS
Artificial Intelligence Working Group (AIWG) AI issues

15. NUMBER OF PAGES
38

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	unlimited

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

1992 Summer SIGAda AIWG Workshop Applications Experiences and Lessons Learned Panel Message From the Panel Chair

M92B0000103
September 1992

Janet Faye Johns

A-1

Contract Sponsor N/A
Contract No. N/A
Project No. N/A
Dept. G033

Approved for public release; distribution unlimited.

MITRE

Bedford, Massachusetts

ABSTRACT

On 24-27 June 1992, the Association for Computing Machinery (ACM) Special Interest Group for Ada (SIGAda) Artificial Intelligence Working Group (AIWG) held a workshop to discuss Ada real-time and AI issues. Panels organized for the workshop focused on blackboard architectures, experiences and lessons learned, real-time development approaches and issues, and Ada 9X issues for AI systems. This paper assesses the state-of-the-art for AI development processes and provides a summary of the related workshop discussions from the experiences and lessons learned panel. Issues associated with requirements analysis, design methodologies, development techniques, test and validation, and maintainability are discussed for AI applications and for AI with Ada applications.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Artificial Intelligence (AI) with Ada is a reality! Based on the information presented at the 1992 Summer Special Interest Group for Ada (SIGAda) Artificial Intelligence Working Group's (AIWG) Applications Experiences and Lessons Learned panel, Ada is a viable language for AI applications. This panel provided a valuable opportunity to accumulate more empirical evidence proving that Ada is being used successfully to implement large-scale AI systems. Congratulations to the panel participants for their AI with Ada successes!

PURPOSE

The AIWG formed the Experiences and Lessons Learned panel to identify problems and further our understanding of the unique issues associated with implementing AI applications with the Ada programming language. During the panel, workshop participants discussed the problems encountered and experiences gained during the development of large AI applications with Ada. This panel provided a vehicle for expressing the AIWG's desires and requirements with respect to the software development process, changes to DoD-STD-2167A, and of course Ada 9X.

Detailed briefings given during the panel discussions are contained in these proceedings. Highlights of the panel discussions and related workshop discussions have been included with the following message from the panel chair. Recommendations for specific AIWG actions are made at the end of this Executive Summary. A workshop participant list is included in Appendix B.

ISSUES

The panel discussed issues covering the complete software life cycle including the challenges of software engineering AI applications; the difficulties with AI requirements analysis; the legacy of an AI requirements analysis; testing, verification, and validation of AI applications; and maintenance for AI applications. Many of the issues and problems discussed during the

workshop are applicable to AI applications developed with any language including the Ada programming language.

Software Engineering

AI applications typically begin with few documented requirements and are defined as well as developed with a series of evolutionary prototypes. This is the current state-of-the-art for AI applications design and development with Ada or any other programming language. Software engineering and Ada design methodologies typically begin with a set of well-defined, testable, verifiable requirements. Therefore, due to the lack of requirements, software engineering is a challenge for AI with Ada developers. Without a set of well-defined requirements, how do you "engineer" AI applications? This question is the subject of on-going debates and research. However, while there are no textbook solutions today, the information in these proceedings describes how AI with Ada practitioners are successfully "engineering" AI applications today.

AI Requirements Analysis

Requirements for AI applications are defined through iteration, that is learning by doing. AI requirements are difficult, if not impossible, to specify without prototypes. Most of the effort for AI projects is spent defining requirements and prototyping. In the case of a 70,000 source line of code Ada application, 2/3 of the calendar time and labor were expended developing prototypes to define the requirements for the AI application.

AI requirements analysis requires a flexible process because defining AI requirements is an evolutionary process. This type of process is not a fixed price problem and the current DoD-STD-2167A may be too rigid for AI applications even with the software engineering discipline provided by the Ada programming language. We discussed many AI specific issues associated with the software engineering process and DoD-STD-2167A. These discussions led us to question whether AI requirements and design are the same as understood in the DoD-STD-2167A and other software engineering environments. Based on these discussions, the AIWG decided to be more active in expressing AI specific requirements and concerns to the standards

bodies that are developing standards which impact the development of AI applications with the Ada programming language.

The legacy of AI Requirements Analysis

Prototypes and human understanding of the problem domain are the standard legacy of an AI requirements analysis effort. What happens to this legacy? We would like to use our human understanding to specify the desired system in clear unambiguous requirements, but this is rarely possible. Prototypes typically reflect our understanding of how to implement a solution and in many cases are required to prove that it is possible to "engineer" the full scale system. Can you successfully "re-engineer" a prototype into a supportable and maintainable system? Based on panel experiences described in these proceedings, Ada prototypes are being re-engineered into supportable and maintainable systems.

Testing, Verification, and Validation

The difficulties inherent in AI requirements analysis inevitably lead to problems with the testing, verification, and validation (V&V) of AI applications. Testing and validation activities ensure that the developed software system satisfies a well-defined set of requirements which unfortunately do not normally exist for AI applications. Verification activities ensure that the developed system is supportable and maintainable which raises issues associated with the feasibility of verifying non-deterministic systems that can learn and adapt over time. Testing and V&V are critical areas of current research because the public and the software engineering community have begun to focus attention on building trusted systems that are correct, dependable, robust, safety critical, efficient, and secure.

The panel discussions regarding testing and V&V led to two interesting thoughts. First, AI applications seek to emulate human intelligence and behavior. How do we test and validate humans? In general, the proof of human intelligence and their ability to learn is through on-the-job performance. Therefore, in essence, humans do not undergo the same scrutiny of test and V&V that we are trying to impose on AI applications. Second, instead of trying to perform an unnatural test and V&V scrutiny of AI applications, perhaps it would be more meaningful to develop techniques and tools that determine if an AI application is fit-for-purpose; that is, does

it match the problem domain, is it operable in the proposed environments, and can it be learned with relative ease.

Maintenance

Maintenance is an area of critical concern as large-scale AI applications are fielded and must be supported for a long life cycle. Knowledge base maintenance is also a critical concern during the iterative development of large knowledge based systems. For an expert system, maintenance traditionally involves changes to the rule base as well as the facts which are normally considered data. The discussions in this area included questions such as "What is knowledge? Are rules considered data or software? Should facts in a knowledge based system be treated as data or software?".

During the panel discussions, several implementation techniques currently being used for expert systems were described as:

1) implementing the rules in Ada for runtime performance

2) use two modes for the rule base: an interpretive mode for rule execution during development and a runtime mode that involves translating the rules to Ada for runtime performance

3) a runtime mixture of interpreted rules and rules implemented in Ada.

One of the developers who is using the first approach of implementing the rules in Ada lamented about the tremendous overhead -- approximately 1 week -- of recompilation for any changes to the 510 rules in his rule base. The selected implementation techniques for an expert systems rule base influence both system maintainability and performance. Based on their development and maintenance experiences, the panel identified a critical need for a support environment that includes a real-time browser for runtime "peeks" into the system and knowledge base maintenance tools.

RECOMMENDATIONS

Software engineering is a challenge for the AI community due to the evolutionary nature of AI applications. Software engineering is one of the strengths and advantages offered by a properly managed Ada environment. We, the AI with Ada community, should develop processes and tools that support the "engineering" of AI applications with Ada. The AIWG should add a section to the AIWG's annual report to describe software engineering processes for AI applications and assess the progress that has been made in the use of software engineering principles to develop AI applications. The AIWG should develop a database cataloging software engineering processes, tools, and applications with periodic publication of this information in Ada Letters.

In order to manage and engineer AI with Ada projects, the AIWG should establish a set of software metrics that are compatible with the evolutionary nature of AI applications. Further, the AIWG should conduct annual surveys to determine the current values for the software metrics so that this information can be used by the Ada community to manage and engineer AI applications. The software metrics should be published in the AIWG's annual report and Ada Letters.

The AIWG should become actively involved with standards bodies that are developing standards and other guidance that impact the development of AI applications with Ada. Of course, the first step in this process is to identify the AI specific requirements that need to be communicated to the standards bodies. The issues and problems discussed at this workshop should be matured into concrete requirements and recommendations for formal submittal to the appropriate standards bodies.

Many of the issues and problems faced by the AI with Ada community are the same problems faced by all AI researchers and developers. The AIWG should work closely with the AI community to concentrate our combined efforts on solving our common problems rather than focusing on the perceived differences between the AI and the Ada communities.

SECTION 1

APPLICATIONS EXPERIENCES AND LESSONS LEARNED PANEL

MESSAGE FROM THE PANEL CHAIR

BACKGROUND

During the past year, I have been collecting information about existing Artificial Intelligence (AI) applications developed with the Ada programming language for the SIGAda Artificial Intelligence Working Group's (AIWG) first annual survey [3]. This was a challenging task for a number of reasons, but I have come to believe that the major impediment to my data collection efforts is the state-of-the-art of developing AI applications. The basic nature of specifying and developing AI systems is a major stumbling block to formulating detailed software metrics and other measures that are so common for "engineered" systems. AI systems generally begin with few documented requirements and are developed with a series of evolutionary prototypes. Typical software metrics are not readily available for AI systems. This fact led me to investigate the current processes used to develop AI applications and the problems of applying current software engineering principles to AI applications.

This briefing was presented at the general session of the Summer '92 SIGAda conference. I presented this material as a devil's advocate challenge that stated the issues in an effort to generate open discussions about the issues. These and many other issues were discussed during the workshop. For the publication of this briefing in the workshop proceedings, I have added relevant information from the workshop discussions to the briefing. My briefing combined with my interpretation of the workshop discussions is the method I have chosen to document the workshop discussions for these proceedings. Appendix A contains my briefing and the workshop discussions.

REFERENCES

1. P. Collard and A. Goforth, November/December 1988, Knowledge Based Systems and Ada: An Overview of the Issues, Ada Letters, pp 72-81.

2. C. Culbert, D. Hamilton, and K. Kelley, 1991, State-of-the-Practice in Knowledge-Based System Verification and Validation, Expert Systems With Applications, Vol. 3, No. 4, pp 403-410.

3. J. Johns, June 1992, 1991 Annual Report for the ACM Special Interest Group for Ada Artificial Intelligence Working Group, MITRE Document M92B0000056.

4. N. Leveson, May 1992, High Pressure Steam Engines and Computer Software, Keynote address at the 14th International Conference on Software Engineering, pp 2-14 of the Proceedings.

5. Xiaofeng Li, September 1991, What's so bad about rule-based programming?, IEEE Software, Vol. 8, No. 5, pp 103-105. Editor's Note: Paul Sanders responded to this article in the January 1992 issue of IEEE Software.

6. D. Woods, April 1992, Space Station Freedom: Embedding AI, AI Expert, Vol. 7, No. 4, pp 32-39.

APPENDIX A

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

This paper assesses the current state-of-the-art for Artificial Intelligence (AI) applications development processes. This paper discusses the human and safety reasons why we need to "engineer" AI applications and presents information about the current processes used to develop knowledge based systems. Issues associated with requirements analysis, design methodologies, development techniques, test and validation, and maintainability are discussed for AI applications and for AI with Ada applications.

This paper uses viewgraphs with an accompanying discussion section that is divided into two parts. The first states current issues while the second part provides a summary of the related workshop discussions.

SIGAda Artificial Intelligence Working Group

Janet Faye Johns

22 June 1992

MITRE

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

AI with Ada: Where are we and where are we going?

Past:

Empirical evidence of past successes

Present:

Common issues face the AI and Ada communities

Future:

Our challenge is the application of sound software engineering principles when we write software to solve our problems.

JFJ 6/92

AI with Ada: Where are we and where are we going?

The 1991 AIWG industry survey [3] provided empirical evidence that the Ada programming language has been used successfully to develop Artificial Intelligence (AI) applications. The difficulty encountered in collecting software metrics for the 1991 AIWG survey raised many questions about the current state-of-the-art for AI requirements analysis, design, and development. We are not alone in our struggle to understand where we have been and where we are going. Recently, the "software crisis" and the emphasis on building trusted systems has caused many of us to examine the state of the entire computing industry.

For the present, many challenges face the AI community that are equally applicable to the AI with Ada community. These include domain specific technical issues; liability issues associated with building trusted systems; the difficulties with requirements analysis for AI systems; design methodologies which adequately encompass rapid prototyping; testing, validation, and verification techniques; maintainability for a long systems life cycle. The AI community and the AI with Ada community need to develop a cooperative effort that can focus our limited resources on developing solutions for our common problems.

Software has come a long way in the past few years and has surpassed hardware in development and maintenance costs. Sound software engineering principles is one proven solution used to control and manage the apparent out of control software development and maintenance costs. For the future, software engineering is a challenge that must be met by both the AI and the AI with Ada communities if we are to manage AI software development efforts and ensure that our systems are both supportable and maintainable for a long life cycle.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Software Safety and Liability Issues

"... The introduction of computers into the control of potentially dangerous devices has led to a growing awareness of the possible contribution of software to serious accidents.

...

Our greatest need now, in terms of future progress rather than short-term coping with current software engineering projects, is not for new languages or tools to implement our inventions but more in-depth understanding of whether our inventions are effective and why or why not.

..."

Nancy Leveson, May 1992, High Pressure Steam Engines and Computer Software Keynote address at the 14th International Conference on Software Engineering, p 2 - 14 of the Proceedings.

JFJ 6/92

Software Safety and Liability Issues

Our ability to "engineer" AI applications is rapidly becoming a deciding factor in the acceptance of safety-critical systems that contain AI applications. After several decades of successful research and prototyping, we are just beginning to grapple with the issues of building "trusted systems" that are safety critical, correct, dependable, robust, efficient, and secure. Other important aspects of building trusted systems that we must consider include understanding the human impact of our AI applications and the liability issues associated with AI applications. The credibility that software engineering provides for a software system will be major factor in our ability to gain support and public acceptance for our AI applications.

How well do we really understand our technology? One of the questions that Nancy asked in her keynote address is: "When software fails, why can't it fail in a safe state?".

You may argue that software can't be unsafe in the sense that software does not physically harm people; but, software now controls hardware that can physically harm people. In our technologically dependent society, embedded software surrounds us in our home, at the office, in our automobiles, and in airplanes. The average 1992 automobile is now estimated to contain more software than NASA used to accomplish the 1967 lunar landing mission. Software in modern aircraft is measured in millions of lines of source code. These facts should be sufficient for all of us to be concerned about software safety.

Articles in magazines and trade journals have begun to discuss legal liability issues such as who is liable and responsible when an AI system provides bad advice or erroneous data.

How many of us could prove that our software will always fail in a safe state?

Are our validation and verification techniques sufficient to prove our software will either never fail or will always fail in a safe state?

How well do we understand the software engineering processes that we use to develop software so that our software is as safe as possible?

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

We must understand the human impact of our inventions.

"... A major airline, known for having the best aircraft maintenance program in the world, a few years ago introduced an expert system to aid their maintenance staff. The quality of maintenance fell. The staff began to depend on the computerized decision making and stopped taking responsibility and making their own decisions. When the software was changed to provide only information and only when requested, quality again rose.

...

The use of computers to enhance safety may actually achieve the opposite effect if the environment in which the computer will be used and the human factors are not carefully considered.

..."

Nancy Leveson, May 1992, High Pressure Steam Engines and Computer Software Keynote address at the 14th International Conference on Software Engineering, p 2 - 14 of the Proceedings.

JFJ 6/92

We must understand the human impact of our inventions.

A common bond among all computer scientists is that we must strive to understand the human impact of our systems. For artificial intelligence (AI) systems that interface with people, we must understand how our systems are actually used. This factor may mean the difference between success or failure for our systems.

In Nancy's example, the underlying "intelligent" system was ineffective when the system made decisions; but, was very effective at providing only solicited advice in response to human requests. This example stresses the importance of the human element in our systems.

How many of us consider these human factors in the design of our AI systems?

How do you specify requirements for these human factors?

How do you measure the correctness of the human factors requirements?

Have we even begun to consider how a human being interacts with and reacts to an AI system that is constantly learning and growing thereby producing unpredictable behavior?

Workshop Discussions

This topic received little attention during the workshop discussions. However, several participants did say that more attention must be paid to the human impact of our AI systems.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Theorem: $\forall$ Artificial Intelligence Problems

$\exists$ An optimal software engineering process

Proof:

Ada promotes good software engineering

...

Ada has been used to solve real world AI problems

...

$\therefore$ Ada provides an optimal software engineering process for all Artificial Intelligence problems.

JFJ 6/92

Theorem: An optimal software engineering process exists for all Artificial Intelligence problems

Our greatest challenge is the application of software engineering principles to the development of AI applications. This theorem and proof are my way of asking the question:

"Are we using sound software engineering principles to develop solutions for our Artificial Intelligence (AI) problems?"

Ada promotes good software engineering practices; but, does Ada support an optimal software engineering process for all AI problems? If not all AI problems, does Ada support an optimal software engineering process for some selected subset of AI problems? What are the issues associated with using Ada to develop AI applications?

Workshop Discussions

An assessment of the experiences of using Ada to develop AI applications shows that many of the issues are equally applicable to all AI development environments. Unfortunately, these issues represent software engineering challenges that span the complete software life cycle:

1) Requirements are difficult, if not impossible, to define until the AI application has been prototyped and developed.

2) Requirements analysis requires rapid prototyping.

3) Design can only be accomplished with iterative prototypes.

4) Test, verification, and validation are difficult and ill-defined for AI software.

5) Little maintenance experience is available for AI applications because few large-scale systems have been fielded.

6) Based on the developer's experiences with prototype knowledge maintenance, knowledge maintenance for a long life cycle is an area of critical concern .

7) Scalability from the requirements analysis and design prototypes to a full scale system is an ill-defined area for AI software.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Our Challenge: Software Engineering with Ada and AI

"One of the most complex systems ever taken into a hostile environment already has projects that leverage AI technology.

...

Ada has been chosen as the Freedom Station language of choice, primarily because of its software engineering characteristics: abstraction, information hiding, modularity, localization, uniformity, completeness, and confirmability.

...

This "design for evolution" is accomplished by conforming to standards (UNIX, FDDI, Ada, X Windows) and designing the interconnections with sufficient bandwidth to allow for faster components as they become qualified for space.

..."

Donald Woods, April 1992, AI Expert, Vol 7, No 4, pp 32-39, <u>Space Station FREEDOM: Embedding AI</u>

JFJ 6/92

Our Challenge: Software Engineering with Ada and AI

The future is bright for both AI and Ada. What does the future hold for AI with Ada? The answer to this question will depend on our ability to use sound software engineering processes in the development of AI systems with Ada -- or any other language.

Workshop Discussions

The 1991 AIWG annual applications survey [3] documented 17 AI with Ada products, eight (47%) of which were less than 5 thousand source lines of code (KSLOC). The AI with Ada applications described in these proceedings and at the workshop represent a sampling of much larger AI with Ada applications with sizes of 40, 45, 70, 153, and 200 KSLOC. Obviously, the future is very bright for AI with Ada applications.

The 1991 AIWG annual applications survey identified the need for software metrics which are virtually nonexistent for AI applications. Participants at the workshop agreed that software metrics are difficult for AI systems and were not surprised about the absence of metrics for AI with Ada applications. Unfortunately, this situation makes it very difficult to manage and cost AI projects.

There were several discussions about projected AI applications for large-scale systems such as the Strategic Defense Initiative (SDI). For large mission-critical systems such as SDI, proven software engineering processes must be developed and matured so that these large AI applications can be managed and accurately budgeted. All of the AI software engineering issues with requirements specification, iterative proof-of-concept design, test and validation, and maintainability will be very critical issues for large-scale embedded SDI applications.

The workshop participants described AI with Ada applications in various phases of the software life cycle. AI with Ada developers are continuing to design and develop applications. However at this point, there is very little experience with operations and maintenance for large-scale AI with Ada systems.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Can we bridge the gap between the Ada and AI cultures?

"... Cross-cultural vantage point. ... On one side is the Ada community that represents the engineer's "how to" culture; and, on the other side, the AI community that represents the scientist's "what" and "why" culture.

...

A fundamental thesis of AI is that "intelligence" can be reasonably approximated by computation; i.e. a computer program.

..."

Phillippe Collard and Andre Goforth, November/December 1988, Knowledge Based Systems and Ada: An Overview of the Issues, Ada Letters, p 72 - 81.

JFJ 6/92

Can we bridge the gap between the AI and Ada cultures?

The representation of knowledge, manipulation of knowledge, reasoning with knowledge, understanding the true meaning of intelligence, and the impact of our systems on intelligent beings are some of the exciting challenges facing all Artificial Intelligence (AI) researchers today. These challenges form a common bond between the AI with Ada community and other AI researchers.

The gap that separates the AI and Ada communities includes many issues associated with our basic problem solving approaches; that is, rapid prototyping versus a software engineering process that begins with a set of well-defined requirements.

Can we develop software engineering processes that encompass the "what" and "why" AI culture?

Workshop Discussions

Rapid prototyping techniques are being used with the Ada programming language. Based on the discussions at the workshop and the AI applications described in these proceedings, the basic AI problem solving approach of rapid prototyping is being used to implement AI applications with the Ada programming language.

The general consensus of the workshop participants is that their AI with Ada projects begin with few well-defined requirements. Therefore, the lack of well-defined requirements is a common characteristic for both AI and AI with Ada applications.

Workshop participants expressed the need for a rich set of AI with Ada tools like those typically found in other AI environments. Several discussions centered around the need for requirements analysis, design, development, maintenance, and real-time analysis tools. These tools could support an environment that approaches the "what" and "why" AI culture while enforcing proven software engineering processes. More research is definitely required to develop tools that support a rich AI software engineering environment.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Requirements Definition with Prototypes

"... Much work is needed to include the concept of rapid prototyping in the software engineering process. Ada's structure is based on the assumption that the design of a software system is derived from a set of crisp requirements. AI applications are not defined so much by crisp requirements but rather by the available knowledge on a particular problem domain that can be integrated in a computer program. To successfully implement AI applications in Ada, one will have to merge these two approaches.
..."

Phillippe Collard and Andre Goforth, November/December 1988, Knowledge Based Systems and Ada: An Overview of the Issues, Ada Letters, p 72 - 81.

JFJ 6/92

Requirements Definition with Prototypes

This article appeared four years ago in the November/December 1988 issue of Ada Letters. How successful have we been with the merger of rapid prototyping with the Ada software engineering process?

Workshop Discussions

Based on the workshop discussions and the AI applications described in these proceedings, rapid prototyping is being used for AI with Ada applications. Although textbook solutions have not been developed, progress has been made in the use of rapid prototyping for AI with Ada applications. All of the AI with Ada applications described in these proceedings used prototyping which provides empirical evidence that prototyping with Ada is not only technically feasible but has become a normal part of the process of developing AI with Ada applications.

The issue of design freedom was briefly discussed at the workshop. Government participants described the situation where they develop prototypes to define requirements and as a consequence develop an implementable design that isn't used by the development contractors because the contractor has design freedom under government acquisition regulations. With the iterative "learning by doing" nature of AI applications, the development contractor frequently repeats the same iterative cycle to design the AI application. Mechanisms to transfer prototype design information to the full-scale engineering development phase will continue to be a topic of discussion.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Maintainability is a big factor in the software life cycle.

"...
If AI and Ada are to coexist, an important element will be the ability for software engineers to understand knowledge engineering problems, and for AI specialists to take into account the requirements imposed by large, long lived projects.
..."

Phillippe Collard and Andre Goforth, November/December 1988, <u>Knowledge Based Systems and Ada: An Overview of the Issues,</u> Ada Letters, p 72 - 81.

JFJ 6/92

Maintainability is a big factor in the software life cycle.

Two-thirds of the life cycle cost a software system is currently being attributed to maintenance and two-thirds of this cost is spent on product improvements and enhancements.

Knowledge and the way we interpret knowledge are very dynamic elements in Artificial Intelligence (AI) systems. Our understanding of knowledge, and how to use it, is constantly changing. For a typical AI system, knowledge -- the knowledge base, the fact base, the inference engine, reasoning model, etc. -- must be maintainable for a long life cycle.

One of the promises of the Ada programming language is that software will be maintainable for a long life cycle. Large, long-lived software projects require that the software be maintainable, supportable, and reliable. This in turn demands a software engineering approach that includes a thorough requirements analysis, a flexible design, and extensive test activities.

How well does Ada support the ever-changing nature of a knowledge based system?

Workshop Discussions

In general, large-scale AI applications have not been operational long enough for much maintenance experience. Software implementation techniques impact maintainability and performance. The AI applications described in these proceedings include three approaches to implementing expert systems:

1) code the rules directly in Ada for runtime performance

2) use two modes for the rule base: an interpretive mode for rule execution during development and a runtime mode that involves generating Ada code for the rules to improve runtime performance

3) a runtime mixture of interpreted rules and rules implemented in Ada.

One of the developers who is using the first approach of implementing the rules in Ada lamented about the tremendous overhead -- approximately 1 week -- of recompilation for any changes to the 510 rules in his rule base.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Are AI with Ada systems testable, maintainable, and reliable?

"...
From a software engineering perspective, rule based programming is a failure because systems developed in it are unmaintainable, untestable, and unreliable.

...
One of the best-known advantages of rule-based programming is that it minimizes the distinction between development and maintenance, because the developer and maintainer deal only with production rules.

...
Maintenance could eventually consume up to 60 percent of the total cost of a product in its entire life cycle. Unfortunately, this advantage is not realized in practice. ..."

Xiaofeng Li, September 1991, What's so bad about rule-based programming?, IEEE Software, Vol 8, No 5, p 103 - 105. Paul Sanders responded to this article in the January 1992 issue of IEEE Software.

JFJ 6/92

Are AI with Ada systems testable, maintainable, and reliable?

The difficulties inherent in AI requirements analysis inevitably lead to problems with the testing, verification, and validation (V&V) of AI applications. Testing and validation activities ensure that the developed software system satisfies a well-defined set of requirements which unfortunately do not normally exist for AI applications. Verification activities ensure that the developed system is supportable and maintainable which raises issues associated with the feasibility of verifying non-deterministic systems that can learn and adapt over time. Testing and V&V are critical areas of current research because the public and the software engineering community have begun to focus attention on building trusted systems that are correct, dependable, robust, safety critical, efficient, and secure.

Workshop Discussions

The panel discussions regarding testing and V&V led to two interesting thoughts. First, AI applications seek to emulate human intelligence and behavior. How do we test and validate humans? In general, the proof of human intelligence and their ability to learn is through on-the-job performance. Therefore, in essence, humans do not undergo the same scrutiny of test and V&V that we are trying to impose on AI applications. Second, instead of trying to perform an unnatural test and V&V scrutiny of AI applications, perhaps it would be more meaningful to develop techniques and tools that determine if an AI application is fit-for-purpose; that is, does it match the problem domain, is it operable in the proposed environments, and can it be learned with relative ease.

Workshop participants identified a critical need for tools that support the full life cycle of AI applications. A real-time browser that would enable developers and maintainers to "peek" into the system was identified as a critical tool for AI applications. In general, the larger the application, the more valuable these tools would become. Problems identified during the tool discussions covered issues such as version walk between the operational system and tools. One suggestion was to develop the tools in Ada and maintain the tools with the AI application. This line of reasoning led to the conclusion that we should be developing a set of useful AI tools and that the AIWG should maintain a database of available tools.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Case Study: Accuracy of Existing Systems

From a survey of 65 AI developers and 5 users:

75% of the systems performed diagnosis
73% of these were in the aerospace industry

The systems were less accurate than expected:

75% were less accurate than expected
- 39% were less than 90% accurate
- 54% were less than 95% accurate

62% were less accurate than an expert

David Hamilton, Keith Kelly, and Chris Culbert, 1991, State-of-the-Practice in Knowledge Based System Verification and Validation, Expert Systems with Applications, Vol 3, No 4, p 403 - 410.

JFJ 6/92

Case Study: Accuracy of Existing Systems

Culbert, Hamilton, and Kelly documented the experiences and problems encountered by knowledge based systems developers and users. In their paper, the terms knowledge based systems and expert systems were used interchangeably by the authors. Although the focus of the authors was verification and validation, the article reveals some very valuable information about the state-of-the-art of knowledge based systems. The article described responses from 65 developers and 5 users. The surveys were distributed to a selected set of users and developers as well as at conferences so that there were a wide variety of survey participants. The authors plan to use the problems uncovered in the survey to make recommendations that will improve the quality of knowledge based systems used for the Space Station Freedom Project.

75% of the surveyed systems performed diagnosis and 73% of these diagnostic systems were used in the aerospace industry. The users and developers in this survey felt that their systems were less accurate than their expectations. Further, 62% were less accurate than a human expert. These facts stress the urgency of addressing software safety and liability issues for AI applications.

As most AI systems are developed without well-defined requirements, how do you quantify accuracy expectations for AI applications?

Workshop Discussions

Accuracy workshop discussions focused on the performance constraints of real-time systems. Basically, an AI system could provide the best possible answer within its timing constraints. After all, this is what humans do, humans provide the best possible answer in the allotted time.

There were no discussions about quantifying accuracy expectations for AI applications. This is obviously an area for future research.

Case Study: Development Without Requirements

52% of the systems had no documented requirements

"...In some cases, requirements were not written because it was felt that a requirements document was a formally written paper that needed to be approved before development could proceed. In other cases, an iterative prototyping development effort took place and was followed by documenting system requirements.
...
There is little understanding of how requirements for an expert system should be generated and documented.
..."

David Hamilton, Keith Kelly, and Chris Culbert, 1991, State-of-the-Practice in Knowledge Based System Verification and Validation, Expert Systems with Applications, Vol 3, No 4, p 403 - 410.

JFJ 6/92

Case Study: Development Without Requirements

52% of the 70 AI systems in the survey had no documented requirements! For those working in the DoD-STD-2167A environment or another modern software engineering environment, this is a frightening statistic because these environments rely on a set of well-defined requirements to manage, scope, and budget software projects. How effective are the processes used to develop AI systems without a set of well-defined requirements? What is even more difficult to understand is how you design, develop, and test AI systems without requirements.

We normally view an Ada development in terms of a set of well-defined requirements. This statistic represents a challenge to the implementation of AI systems with Ada. We know that Ada is being used to develop AI systems. Are Ada developers using the same techniques that are being used in other AI environments and languages?

Workshop Discussions

Based on the information presented at the AIWG workshop, Ada developers are developing AI systems with few well-documented requirements. Ada developers are also using iterative, rapid prototyping techniques similar to those used in other AI environments. The projects described in these proceedings used prototyping techniques to perform requirements analysis as well as to iteratively design and develop AI with Ada applications.

Requirements for AI applications are defined through iteration, that is learning by doing. AI requirements are difficult, if not impossible, to specify without prototypes. Most of the effort for AI projects is spent defining requirements and prototyping. In the case of the 70,000 source line of code (SLOC) Ada application described in these proceedings, 2/3 of the calendar time and labor were expended developing prototypes to define the requirements for the AI application.

AI requirements analysis requires a flexible process because defining AI requirements is an evolutionary process. This type of process is not a fixed price problem and the current DoD-STD-2167A may be too rigid for AI applications even with the software engineering discipline provided by the Ada programming language. We discussed many AI specific issues associated with the software engineering process and DoD-STD-2167A. These discussions led us to question whether AI requirements and design are the same as understood in the DoD-STD-2167A and other software engineering environments. Based on these discussions, the AIWG decided to be more active in expressing AI specific requirements and concerns to the standards bodies that are developing standards which impact the development of AI applications with the Ada programming language.

Case Study: Design Methodologies

"...
40% used a cyclic model for development. However,
22 % of the respondents stated that no model was followed.
..."

```
Loop    -- cyclic development model
           define requirements
           design
           rule generation
           prototype
End loop
```

David Hamilton, Keith Kelly, and Chris Culbert, 1991,
State-of-the-Practice in Knowledge Based System Verification and Validation,
Expert Systems with Applications, Vol 3, No 4, p 403 - 410.

Case Study: Design Methodologies

Many of the survey respondents did not use a formal design and development methodology. What are the design methodologies used to develop AI with Ada applications?

How well do these design methodologies support a long system life cycle for a constantly changing knowledge based system?

How well do the Ada software engineering characteristics of abstraction, information hiding, confirmability, etc. support an ever-changing knowledge based system?

Workshop Discussions

Based on the workshop discussions, AI with Ada developers are using a variety of design techniques and all of these techniques involve prototyping.

Workshop participants identified a critical need for tools that support the complete software life cycle in addition to an iterative design process. There was general agreement that the evolutionary nature of AI applications requires prototyping and a spiral or cyclic design process.

There were several discussions about the use of fourth generation languages (4GLs), language translators, and code generators. Should 4GLs or language translators be allowed in an Ada development? If translators are allowed, what is maintained -- the source language or the generated Ada? If the maintainer does not have the same tools as the developers then all maintenance advantages will be lost. How do you ensure that the language translation scenario will be usable and supportable for a long life cycle? There was general agreement that translation and code generation tools could be written in Ada and maintained as part of the Ada software to ensure that the tools had the same life cycle as the generated software.

SIGAda Artificial Intelligence Working Group
Summer '92 SIGAda Workshop

Case Study: Operational Prototypes

"...
Although we attempted to get respondents to state that their system was either 'a prototype' or 'operational', we received indications that the distinction was often difficult to make.
..."

David Hamilton, Keith Kelly, and Chris Culbert, 1991,
State-of-the-Practice in Knowledge Based System Verification and Validation,
Expert Systems with Applications, Vol 3, No 4, p 403 - 410.

Case Study: Operational Prototypes

As discussed in the survey, 70% of the systems were operational and the rest were considered prototypes but some of these prototypes had operational users. The respondents could not clearly distinguish between prototypes and operational systems -- their systems were operational prototypes. How does this "reality" fit with the Ada design and development methodologies?

What are the software engineering implications of this statistic with respect to:

-- requirements analysis
-- design and development
-- test methods
-- verification and validation
-- configuration management
-- operations and maintenance?

Workshop Discussions

Will your prototype ever grow up and leave home? Prototypes tend to live long beyond the AI requirements analysis and proof-of-concept design phases of the systems life cycle. Although these proceedings document prototypes that have been successfully transitioned into operational systems, workshop discussions reveal that more research is necessary to develop processes that support this transition -- assuming that this transition is desirable. In some of the workshop discussions where participants described their operational prototypes it was apparent that the developers have maintained a very close relationship with the operational users and the developers were still responsible for the prototypes.

Prototypes and human understanding of the problem domain are the standard legacy of an AI requirements analysis effort. What happens to this legacy? We would like to use our human understanding to specify the desired system in clear unambiguous requirements, but this is rarely possible. Prototypes typically reflect our understanding of how to implement a solution and in many cases are required to prove that it is possible to "engineer" the full scale system. Can you successfully "re-engineer" a prototype into a supportable and maintainable system? Based on panel experiences described in these proceedings, Ada prototypes are being re-engineered into supportable and maintainable systems.

Appendix B

AIWG '92 Workshop Participants

Henry Baker
Nimble Computer
16231 Meadow Ridge Way
Encino, CA 91436
818-501-4956 (phone)
818-986-1360 (FAX)

James Baldo Jr.
Institute for Defense Analysis
1801 N. Beauregard St
Alexandria, VA 22311-1772
703-845-6624 (phone)
703-845-6848 (FAX)
baldo@ida.org

Mark R. Bowyer
DRA Farnborough Q153
Hants GU14 6TD, UK
252-24461 x2503 (phone)
252-377247 (FAX)
r_bowyer@uk.mod.hermes

Jorge L. Diaz-Herrera
SEI CMU (MSE project)
4500 Fifth Ave
Pittsburgh, PA 15213-3890
412-268-7636 (voice)
412-268-5758 (FAX)
jldh@sei.cmu.edu
Rob Ensey

Boeing Computer Services
Box 24346, Mailstop 9H-84
Seattle, WA 98124
206-394-3055 (phone)
206-394-3064 (FAX)
rob@patty.amas.ds.boeing.com

John Haun
DRA Portsdown
Portsmouth Hants P06 4AA
Hampshire, UK
0705-333839

Rich Hillard
Intermetrics
Cambridge, MA 08138
617-661-1840
rh@inmet.com

Janet Johns
MITRE Mailstop K203
202 Burlington Road
Bedford, MA 01730
617-271-8206 (phone)
617-271-2753 (FAX)
jfjohns@mitre.org

Mark Johnson
SMC/CNWS
Los Angeles AFB, CA 90009-2960
310-363-8770 (phone)
310-363-8725 (FAX)

Michael Looney
DRA Portsdown
Portsmouth Hants P06 4AA
Hampshire, UK
44-0705-332330 (phone)
44-0705-333543 (FAX)
MJL%ARE-PN.MOD.UK@Relay.MOD-UK

John Miles
DRA - Maritime Division
ARE Portsdown
Portsmouth Hampshire
P064AA England
0705-333839

Howard E. Neely III
Hughes Research Laboratories
3011 Malibu Canyon Rd
Mailstop MA/254/RL69
Malibu, CA 90265
310-317-5606 (phone)
310-317-5484 (FAX)
NEELY@MAXWELL.HRL.HAC.COM

John Tunnicliffe
DRA Portsdown
Portsmouth Hants P06 4AA
Hampshire, UK
0705-332002 (phone)
0705-333543 (FAX)

George (Rick) Wilbur
Boeing Computer Services
Box 24346, Mailstop 9H-84
Seattle, WA 98124
206-394-3055 (phone)
206-394-3064 (FAX)

Mik Yen
Boeing Defense & Space Group
Box 3707, MS4C-63
Seattle, WA 98124-2207
206-662-0213 (phone)
206-662-0115 (FAX)